# Beyond Clients and Servers

Erik van Mulligen[1,2], Teun Timmers[1]
[1]Department of Medical Informatics, Erasmus University Rotterdam, The Netherlands
[2]University Hospital Dijkzigt Rotterdam, The Netherlands

*Computer scientists working in medical informatics have to face the problem that software offered by industry is more and more adopted for clinical use by medical professionals. A new challenge arises of how to combine commercial solutions with typical medical software that already exists for some years and proved to be reliable with these off-the-shelf solutions [1]. With the HERMES project, this new challenge was accepted and possible solutions to integrate existing legacy systems with state-of-the-art commercial solutions have been investigated. After a period of prototyping to assess possible alternative solutions, a system based on an indirect client-server model was implemented with help of the industry. In this paper, its architecture is described together with the most important features currently covered. Based on the HERMES architecture, both systems for clinical data analysis and patient care (cardiology) are currently developed.*

## INTRODUCTION

After three decades of medical informatics and a large amount of local solutions available, the question arises whether a new approach might bring a solution to provide clinicians with the best possible computer support. This new approach could try to combine the current power of commercially available applications with the medical legacy systems into a medical workstation.

Exactly this was the main objective of the HERMES project when it was started about six years ago [2]. From a computer scientific point of view, the question arose whether it was technically possible to encapsulate existing software, both commercial and local developments without modification and make their functions accessible as a reusable component. An architecture that supports the integration of reusable components into medical workstations for various medical tasks and a user friendly interface through which the components can be addressed were the other two main objectives for the HERMES project.

The project started with the development of a prototype called MW2000, with which the technical feasibility of such an approach was tested [3]. The available computer power, network technology and ability of graphical user interfaces were assessed as being adequate, stable and mature enough for a medical workstation. From a user evaluation of the prototype MW2000, it was concluded that such an approach really can help clinical professionals with their task. With the help of industry, a HERMES consortium has formed that developed from experience with the prototype an open integration architecture, and based on that architecture medical workstations for clinical data analysis, for the outpatient clinic for patients suffering from heart failure and for occupational health care have been developed [4].

A project that addresses similar research questions is the HELIOS project [5]. Although their prime focus has not been the encapsulation of existing sofware in reusable components, but the creation of a software engineering environment that supports the structured development of new components, the technical mechanism is comparable. Through a software bus, components can activate functions resident in other components. Integration in this project is restricted to components that have been developed according within the HELIOS environment and integration of existing software did not receive much attention.

Integration architectures based on reusable components have been proposed by others. Greenes promoted this practice as a mechanism to build complex applications and reach a critical mass of functionality [6]. Outside medical informatics, the Object Management Group (OMG) proposed a Common Object Request Broker Architecture (CORBA), consisting of a common object repository [7]. In their view, each object has methods through which components can be accessed and data can be exchanged. Currently, a first implementation of CORBA is available. Other initiatives, such as Distributed Computing Environment (DCE) and Open Network Communication (ONC) Remote Procedure Calling (RPC) are less ambitious and mainly focus on transparent network-wide procedure calls (no brokering or encapsulation mechanism) [8].

0195-4210/94/$5.00 

CORBA, DCE, and RPC bind a call and the actual procedure at compile time. In HERMES, we use very late binding; at execution time, a generic call is delivered to a central broker, bound by this broker to a specific procedure call and finally this call is forwarded to a service and executed. With this feature, it is possible to very easily switch between alternative components that provide the requested functionality and (2) allows the broker to evaluate different break-downs of a generic call for particular criteria (such as speed, CPU-cost, reliability etc.).

In many research projects integration is limited to data. Projects such as IAIMS and Hewlett Packard's Physicians Workstation primarily aim at combining data from different (heterogeneous) sources [9,10]. Research topics in this direction are related to data semantics, modeling and communication. Earlier work in the field of integration investigated other communication mechanisms between the processes and sources involved in the integration [11,12].

In this paper, we will extend on the basic HERMES methodology that turns existing legacy systems into open reusable services and elaborate on those features that have proven to be esential for such an approach in the medical domain.

## METHODOLOGY

### Indirect client-server model

In the traditional client-server model, a client composes a request and forwards it to a server; the server on its turn will compose a reply and send it back to the client. In this model, the client selects the service (and host) that can solve the request. Furthermore, both client and server have to agree on the syntax and semantics of the message language in which the request and reply are expressed; any change must be implemented in both client and server. One solution to this is to reduce the mutual dependence between client and server by having a mediator that establishes indirect client-server communication. The task of this mediator is to do some interpretation and transformation of the requests that are exchanged between clients and servers.

HERMES utilizes as mediator a brokering mechanism that dynamically searches for a binding between a request of the client and the available procedures and data of the services. An overview of this mechanism is shown in Figure 1. This brokering mechanism is supported by a broker that reads an object-oriented database with all information about its environment: there are classes of requests with request instances, in the service classes instances identify the services that are available in the network, and a host class contains an instance for each host. Mandatory parameters for the various requests are specified as instances in parameter classes. Furthermore, a user is represented by an instance in one of the user classes. Relations between all these instances contain the operational knowledge of what service can handle what request, what mandatory parameters should be included in a request, on what host a service is resident and what preferred services are specified for each user.

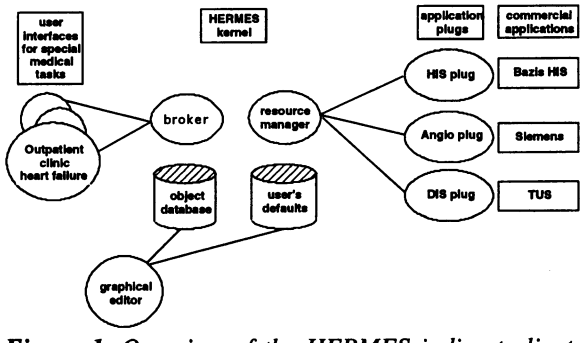


*Figure 1. Overview of the HERMES indirect client-server architecture.*

New requests, parameters, services, hosts, and users can be made operational by creating objects for them in an object database. Relations between objects specify the various bindings that have to be made; e.g. the relation 'can_be_handled_by' between a request and a service object defines that that service can handle that request. Through this broker, clients can directly benefit from newly installed services and hosts, without having to change the client. Furthermore, differences between the mandatory parameters for services that supply similar procedures are solved by the broker by adding these. The current brokering algorithm selects the first from a user-defined list of alternative services.

### Resource manager

Although the client-server model offers an elegant mechanism to separate functionality into reusable services that can be shared by different clients, the disadvantage of all these separate processes is a significant increase of both computer load and memory use. From experience with the prototype integrated workstation, we learned that efficient resource management is essential to improve the

throughput of systems that use a brokered client-server mechanism. Therefore, a resource manager has been inserted between the client and server communication that attempts to combine per user requests to already open service sessions. This strategy reduces the initial time to start a service and initialize a context. In addition, the resource manager will add user preferred parameters to requests and overrule the value of parameters set by the broker.

**Message Language**

Requests are composed by a client and follow the HERMES message language syntax. In this message language, all dependencies on the host's hardware and on the client process are avoided so that it can be passed between any host and process. For the transport and data level of the communication, a commercially available product has been selected that is supported on many platforms (Berkeley sockets).

The message language uses tags to identify the various parameters in a message. There is a set of predefined tags that are specified in all requests. Special constructs are added to include lists and objects (C-structures), and to automatically encrypt and decrypt data. One important feature of the HERMES kernel is that parameters do not necessarily have to be explicitly defined in the HERMES kernel. This allows dynamically construction of new parameters, thus freeing the kernel from having to know all possible parameters in advance. A special feature is the raw data type that can be used to pass data that does not follow the syntax; with this envelope feature 'foreign' messages (such as HL7) can be passed without having the broker and resource manager interpret this data for obedience to the syntax.

**HERMES library**

To free the developer from knowing how to express internal data structures and variables in the HERMES message language and to allow modifications or replacement of the message language by newly developed standard message languages (such as ASN-1), a special library has been provided. Both the client and the service use this library provided with HERMES as their stubs that translate between the internal data and request representation and the HERMES message language. The library consists of a set of high-level functions through which a client can, e.g., compose and send, and identify internal data that should be included in the request. Most importantly, we extended the stub's functionality so that it automatically includes parameters that are essential for the client's identification (hostname, username, process number, etc.). For the server, the library provides procedures that parse the message and construct an internal representation of its contents.

**Callback mechanism**

The HERMES library contains procedures that facilitate true asynchronous communication by installing a callback; this callback is automatically executed by the stub when an answer is received by the client. The server can use the same mechanism to install a callback for receiving requests. The stubs also handle several internal requests that are composed by the broker and resource manager to keep track of the status of clients and servers. Exception handling, connection termination, session management, and login handling are automatically provided by the stubs.

The HERMES library has been optimized for transmission of large data volumes (series of images or multi-record data for clinical research). In that case, data is automatically stored on a file to avoid internal memory problems.

**Application Programming Interfaces**

Developers can use the HERMES library to write macro procedures that allows them to define a procedure that composes a request for a service. The arguments of the procedure are passed as data for the request to the service. A set of these procedures forms the Application Programming Interface (API) for a service. These procedures are made available in libraries that can be linked with clients (see Figure 2).

**Independence**

One of the most important features of the HERMES integration architecture is its layered design that allows new commercial solutions that provide similar functionality to be easily put in. First of all, the broker mechanism allows a client to be truely independent from a service. Secondly, the HERMES library allows both client and server to use procedures without having to know the message syntax. The HERMES library procedures can be modified to generate messages according to a standard message language when available. Apart from this, client and servers can be developed separately, using different programming languages and operating systems.

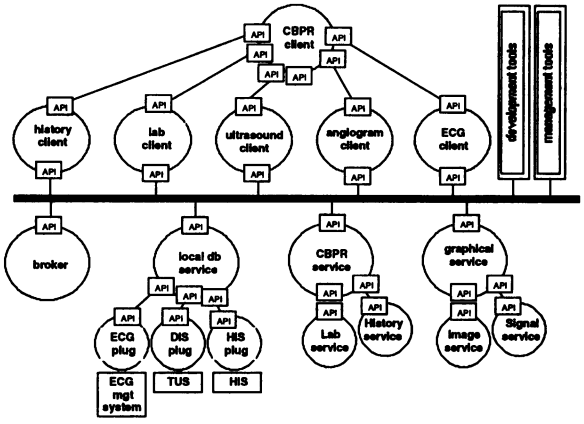


*Figure 2. Network of clients and services communicating through HERMES APIs which provide easy access to data and functions.*

**Encapsulation**

Another key feature of the HERMES integration architecture is its ability to embed legacy systems without modifications in a plug. This plug behaves to HERMES as a network-accessible true service, but translates incoming requests to the data format and instructions for the underlying system. These instructions can be either formulated as a batch for systems that accept batch processing or as a series of keystrokes for interactive applications. For integration of PC software, the plug can be enhanced with an additional layer between the plug and software that emulates a MS-DOS PC (commercially available).

**Sessions**

The notion of a session is important when integrating existing systems. Many of these systems are stateful, i.e. the meaning of an action is dependent on previous actions, and requires preservation of a context. Although a connectionless approach (with no notion of sessions) requires less resources and is much easier to realize in a network, it lacks this essential feature for encapsulation of existing systems. In addition, when clients share stateful services they can exchange information through the created context and avoid redundant interaction with the user to recreate the context. Of course, connectionless communication can be simulated in HERMES by shutting a session directly after creation. Note that the resource manager incorporates functionality to reduce the resource load of a session oriented communication mechanism by reusing open connections.

**Data model server**

All data available in the network is described in a data model. This description includes meta-data about the type, the range, the coding, the medical name, the size, keys, etc. and (medical) relationships with other data. This data model is supported by a data model server that offers other components (clients and services) the possibility to query for information. The data model contains per object a quadruple (dbms, database, relation, attribute) that exactly identifies where the data is located.

The data model server allows various conceptual views on the data. These views can be created by medical professionals and contain medical relationships between the data. The end nodes of this conceptual tree are links to attribute elements in the data model. These views can be modified independently of the underlying data model (and consequently database schema).

**User-interface**

All interaction with the user is window-oriented. In HERMES, the user-interface has been implemented with X11. Although most interaction with the user is handled by a client, it is not exclusively restricted to a client. Therefore, it is essential that the window systems, such as X11, enables remote services to interact with the user on the local host.

Secondly, in HERMES clients can be composed of subclients. To allow management of the windows of the subclients, their user-interfaces have to be integrated as part of the client's user-interface. This feature, called reparenting, is supported by X11 and included as a special option in the message syntax. The HERMES stubs contain code to automatically perform user-interface reparenting when this tag is included in the message.

## ASSESSMENT

**Flexibility and extensibility**

The very late binding of a request to the actual call to a procedure in a service has proven to be a very powerful feature in the development of integrated medical workstations. Already during the development, similar functionality was offered by different applications. With one single client, we were able to switch between different services without having to rewrite a single line of code. Furthermore, it proved to be powerful for version control; depending on the user, different versions of a service can be accessed by the same client. New

functionality could also be easily added by editing the broker's database with a graphical editor. However, no checks were included to preserve the correctness of the name space (i.e. duplicate requests can coexist). To improve the management of requests, a special editor has to be developed that ensures consistency.

### Plugs

We created several plugs for batch-oriented systems and interactive systems with and without PC emulation. It appeared that writing a plug is not a trivial task and that only little of the code is generic. We divide a plug into three main modules: (1) translation of the data in the HERMES format to the application format, (2) generation of instructions for the application from the request, and (3) transformation of the application's output to the HERMES format. Code for step (1) can be generated more or less automatically. With a macro recorder interactive applications can be automated in step (2); for batch-oriented applications more programming is required. Step (3), transformation of the output is typically a manual programming job and little automated support can be given.

## CONCLUSIONS

For introduction of open system technology and distributed processing in a medical environment, it is essential to provide an architecture that is able to cope with existing systems (both legacy information systems and systems from different vendors) without having to modify these systems. Furthermore, attention should be paid to develop a maximally flexible approach that allows systems to be replaced with new, preferably suited for operation in a client-server environment services without having to change all dependent applications/clients. A broker that binds requests to services and manipulates messages seems a suitable approach to this problem.

The price to be paid for a client-server approach is an increase in processor and memory load, which can be minimized by introducing a special server that monitors all running services and tries to combine requests to a single server process. The reusability of the server components in HERMES favours the development of medical applications for specific medical tasks rather than a generic application for various tasks. Our experience with building an environment for clinical data analysis and for the outpatient clinic for patients suffering from heart failure supports this.

HERMES offers an effective path that allows for a more evolutionary than revolutionary approach to integration. Compared with other efforts in this direction, its capability to embed legacy systems in an open environment favors its use in domains with a large and rapidly changing installed base of software (typical for the medical domain). The price to be paid for this is some computational overhead, which however can be compensated by intelligent resource management. Our experience is that the HERMES approach allows development of reusable components that can be shared by various applications for patient care and clinical data analysis.

### Acknowledgement

Hewlett Packard Medical Products Group Geneva is greatly acknowledged for their financial support.

### Reference

[1] Power LR. Post-facto integration technology: new discipline for an old practice. In: Hg PA, Ramamoorthy CV, Seifert LC, Yeh RT, eds. *Proceedings of the First Conference on Systems Integration.* IEEE Computer Society Press, 1990:4-13

[2] Van Mulligen EM, Timmers T, Van Bemmel JH. A new architecture for integration of heterogeneous software components. Methods of Information in Medicine 1993;32:292-301

[3] Van Mulligen EM, Timmers T, Van den Heuvel F, Van Bemmel JH. A prototype integrated medical workstation environment. Computers Methods and Programs in Biomedicine 1993:39:331-341

[4] Cornet R, Van Mulligen EM, Timmers T, A cooperative model for integration in a cradiology outpatient clinic. Accepted for SCAMC'94.

[5] Degoulet FJ, Coignard J, Scherrer JR et al.. The Helios European Project on Software Engineering. In: Timmers T, Blum BI (eds.) Software Engineering in Medical Informatics. Amsterdam: Elsevier Scientific Publishers. 1991:125-37

[6] Greenes RA. Promoting productivity by propagating the practice of "plug-compatible" programming. In: Miller RA, ed. *Proceedings of the 14th Annual Symposium on Computer Applications in Medical Care. Washington DC.* New York: IEEE Computer Society Press 1990:22-6

[7] OMG. The Common Object Request Broker: Architecture and Specification. OMG Document 91.12.1, X/Open publisher

[8] OSF. Introduction to OSF DCE. Englewood Cliffs, New Jersey: Prentice Hall, 1992.

[9] Young CY, Tang PC, Annevelink J. An Open Systems Architecture for Development of a Physician's Workstation. In: Clayton PD, ed. *Proceedings of the 15th Annual Symposium on Computer Applications in Medical Care.* Washington DC. New York: McGraw-Hill Inc. 1991:491-5

[10] Roderer NK, Clayton PD. IAIMS at Columbia-Presbyterian Medical Center: accomplishments and challenges. Bull Med Libr Assoc 1992;80:253-62.

[11] Sinha A. Client-server computing. Communications of the ACM 35, 7 (1992):77-97

[12] Wiederhold G. Views, objects and databases. IEEE Computer, 19;12:37-44